design ideas

Edited by Bill Travis and Anne Watson Swager

Circuit detects first event

Kelly Flaherty, National Semiconductor, San Mateo, CA

THE CIRCUIT IN **Figure 1** is a "first-event" indicator, like a game show's "who's first to answer" detector. It indicates which of the two momentary switches, S_1 or S_2, closes first by latching the corresponding channel, IC_A or IC_B, to a high state. As either of the outputs latches high and lights its respective LED, it locks out the other channel and prevents it from triggering. The other momentary switch, S_3, resets either of the latched outputs to its initial low (LED-off) state. At the initial condition, the positive input of each comparator is approximately at 0V, because both outputs are low. The negative inputs are at $V_{BAT}/11$, as set by voltage divider R_4 and R_5. In this initial condition, assume that S_1 is momentarily closed. The positive input of IC_A becomes $^5/_6(V_{BAT})$, as set by voltage divider R_{2A} and R_{3A}. Because $^5/_6(V_{BAT})$ is greater than VBAT/11, the output of IC_A goes high, and the positive input of IC_A latches its threshold to approximately $V_{BAT}/6$.

Correspondingly, the negative input of IC_B latches to approximately V_{BAT}, thus preventing S_2 from triggering IC_B's output high. S_3 resets (turns off) either of the active outputs by pulling the inverting inputs one diode drop below V_{BAT}. Both channels are then in their initial condition and ready to go again. The LMC6762 dual comparator is a good fit for this application, because it draws only 7-µA quiescent current, and it has rail-to-rail inputs and outputs. The comparator's sourcing capability allows it to easily drive an LED. **Figure 2** shows how you can cascade two first-event detectors to obtain more channels.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

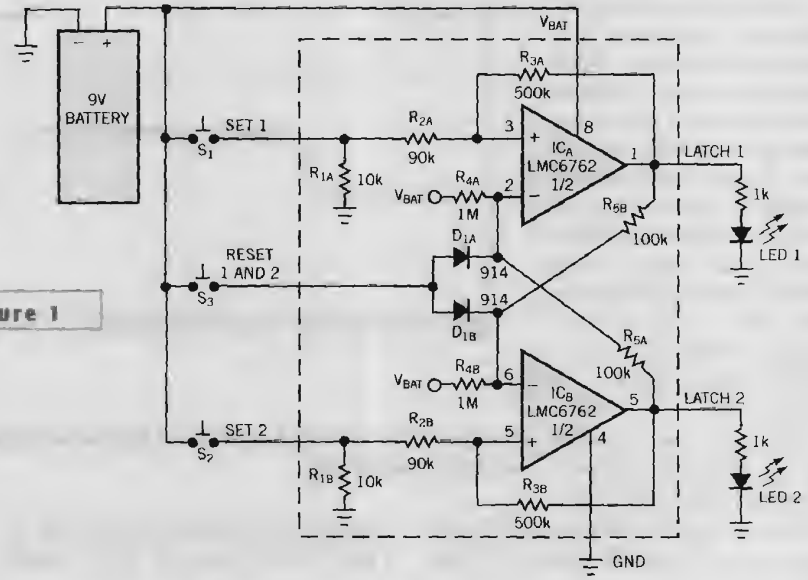


Figure 1

Which button did you push first? This circuit reveals the answer.

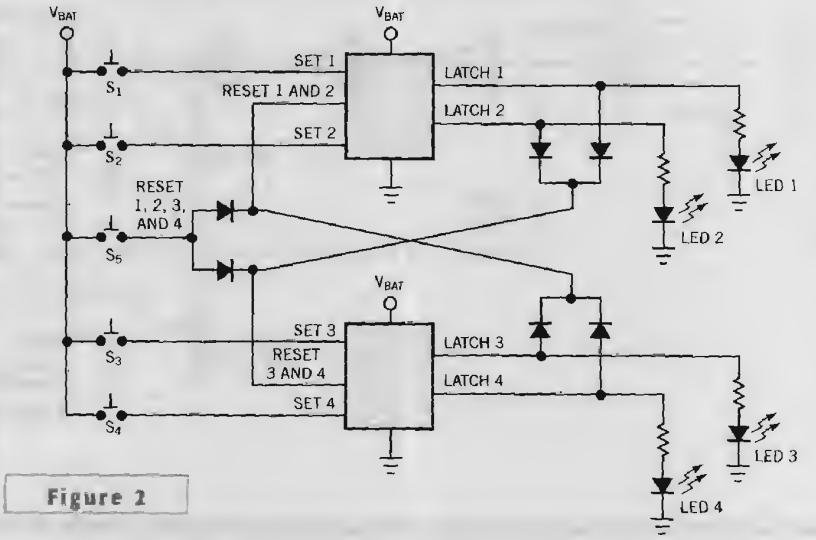


Figure 2

With the addition of six diodes, you can cascade two first-event detectors.

High-speed pulse generator has programmable levels

John Guy, Maxim Integrated Products, Sunnyvale, CA

Lilliputian dimensions associated with the submicron geometries of most digital and many analog processes result in much faster circuit operation. As ICs speed up, the rise and fall times of most pulse and function generators, which are typically 5 nsec, become inadequate for measuring time intervals lower than 20 nsec. You can overcome this limitation with analog comparators or advanced CMOS logic gates, which create faster digital edges. Their rise and fall times are fast enough, but the signal levels include ground and V_{CC} only.

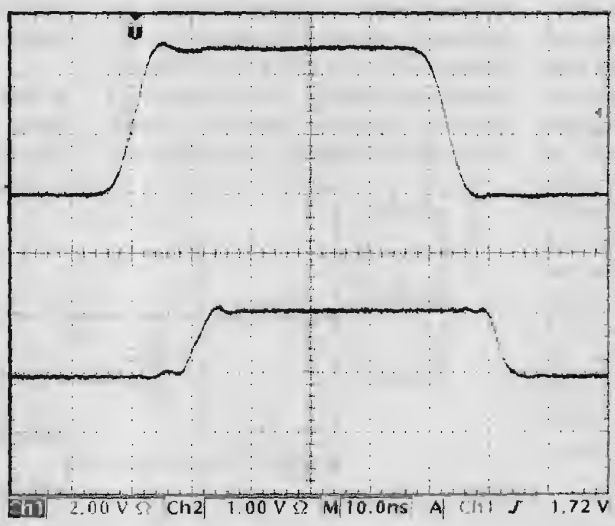


Figure 2

The input (lower) and output (upper) traces illustrate fast output transitions and settable output levels.

Designers have applied the submicron processes that high-speed digital circuits use to analog switches as well, so the turn-on and turn-off times for these switches also produce fast rise and fall times. What's more, an SPDT (single-pole, double-throw) switch can create pulses whose high and low levels are programmable.

A feature of analog switches that hinders their use as pulse generators is the intrinsic built-in delay—the break-before-make time—that guarantees that an SPDT switch does not short the two switched terminals together during a transition. Unfortunately, this delay and the switches' finite turn-on time also extends the rise and fall times. You can avoid this effect by adding a dynamic pullup and a dynamic pulldown to the circuit (**Figure 1**). A sufficiently low pullup and pulldown impedance can drastically improve the corresponding rise and fall times.

The input clock signal, Φ_1, controls an

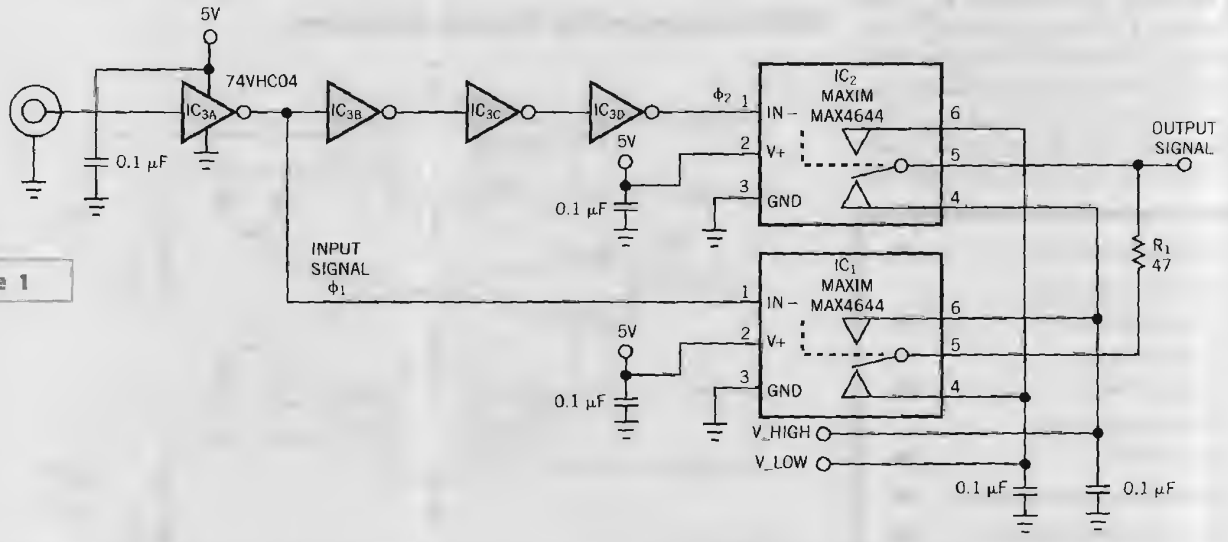


Figure 1

Analog switches provide dynamic pull-up and pull-down at the output of this pulse generator to ensure fast rise and fall times.

SPDT analog switch, IC_1, which the circuit configures as the pullup/pulldown driver. The input clock signal also drives a high-speed CMOS inverter, IC_3, to create a delayed clock signal, Φ_2. The delayed clock drives an SPDT analog switch, IC_2, which the circuit configures as the output driver.

Consider the steady-state condition in which Φ_1 is low and Φ_2 is high. IC_1's COM pin and IC_2's COM pin connect to V_LOW, and a rising edge on Φ_1 causes IC_1 to pull the output signal high. Because the series resistor, R_1, is large with respect to the MAX4644's on-resistance, or 47Ω versus 2.5Ω typical, the immediate effect on output voltage is minimal. However, when Φ_1 propagates through the inverter string, the falling edge of Φ_2 causes IC_2 to transition from V_LOW to V_HIGH. The presence of a low-impedance pullup, R_1, provides drive for the signal transition, and the closing of IC_2 quickly follows.

The input signal is 5V logic, and the output swings from 1V to 2V (**Figure 2**). You can set V_LOW and V_HIGH to any level within the supply range for IC_1 and IC_2. The circuit's quiescent current is essentially zero, with brief peaks only during the output transitions. Rise and fall times at the output are approximately 4 nsec, and the output impedance is 2.5Ω.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Low-battery indicator has high efficiency

Joe Neubauer, Maxim Integrated Products, Sunnyvale, CA

THE USUAL METHOD for implementing the low-battery warning featured in most battery-operated equipment is to illuminate an LED. However, the LED exacerbates the low-battery condition. You can greatly reduce the LED's power consumption by operating it at a low frequency and a low duty cycle. An existing LBO (low-battery output) like that found in dc/dc converters offers a convenient way to light the LED (**Figure 1**). IC_1 is a small, inexpensive comparator with shutdown capability, housed in a six-pin SC70 package. It remains in shutdown condition while the battery is at normal operating levels but asserts LBO when the battery voltage falls below a preset threshold. Active-high LBO is usable as shown, but an active-low warning, $\overline{LBO}$, requires the optional circuitry shown. IC_1 turns on, causing the LED to flash according to the following analysis: First, you want to keep the duty cycle low: $DC=t_{ON}/(t_{ON}+t_{OFF})$. You derive the on-time from the equation for time-varying voltage across a charging capacitor: $V(t)=V(1-e^{-t/RC})$, so $t_{ON}=-R_5Cln(_{TRIPHI}1-V_{TRIPHI}/V_{OUT})$. You then derive the off-time from the equation for time-varying voltage across a discharging capacitor: $V(t)=Ve^{-t/RC}$, so $t_{OFF}=-R_4Cln(V_{TRIPLO}/V_{OUT})$. Use Kirchoff's current laws to find the comparator's high and low trip levels: $V_{TRIPHI}=V_{OUT}\{R_3(R_1+R_2)\}/[R_3(R_1+R_2)+R_1R_2]$, and $V_{TRIPLO}=V_{OUT}[R_3R_2]/[R_3(R_1+R_2)+R_1R_2]$. Assuming a 2.5% duty cycle and assuming that the LBO trips the comparator on when the battery voltage equals 3V, the resulting trip levels are 1V for low and 2V for high. The standard component values corresponding to this performance are: $C_1=0.1\ \mu F$, $R_1=R_2=R_3=1\ M\Omega$, $R_4=3.6\ M\Omega$, and $R_5=91\ k\Omega$.

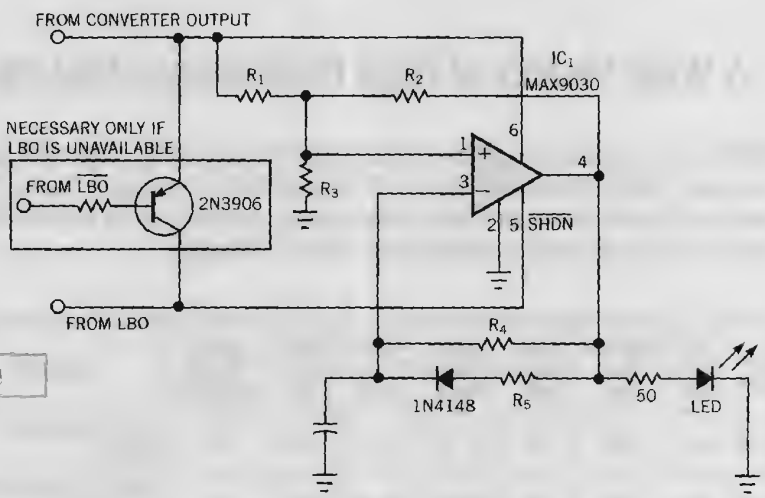


Figure 1

Operating the low-battery LED at low duty cycle saves power and extends battery life.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

LCD-bias supply provides precise tracking

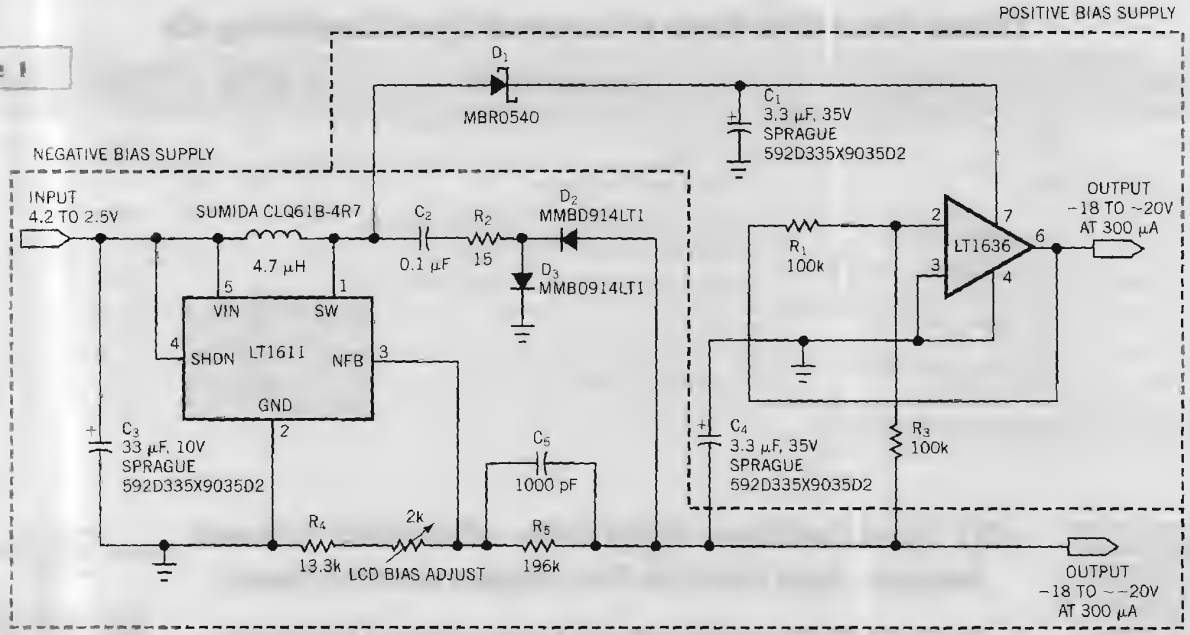
David Kim, Linear Technology Corp, Milpitas, CA

SMALL MONOCHROME LCD systems often require split (dual)-bias supplies with precise voltage tracking to prevent plating of the LCD. The circuit in **Figure 1** provides ±18 to ±20V adjustable LCD bias voltages with 1% tracking accuracy. The circuit operates from a single 4.2 to 2.5V Li-ion cell for portable monochrome LCD applications. The circuit comprises two blocks: a negative-bias supply using the LT1611 inverting switching regulator and a positive-bias supply using the LT1636 rail-to-rail op amp. The LT1611 converts the Li-ion battery input voltage to a negative output voltage. The combination of 1.4-MHz switching frequency and a 36V internal switch results in small, low-profile circuit. LCD bias requires high voltage at low current. A charge pump consisting of C_2, C_4, D_2, and D_3 generates the negative output voltage. Some benefits of this circuit topology include zero output power during shutdown and low output ripple.

The LT1636 rail-to-rail op amp generates the positive LCD-bias output. The large capacitive-load capability, low quiescent current, and high-impedance input stage make the LT1636 suitable in this application. The LT1636 inverts the LT1611's output to provide the positive LCD-bias voltage. To meet the 1% tracking requirement, you should use a precision-resistor network, such as the 664 series from BI Technologies (www.bitechnologies.com), for R_1 and R_3. The unique input stage of the LT1636 allows you to generate the V_{CC} of the inverting op amp from the rectified switching waveform (using D_1 and C_1) of the LT1611 switching regulator. You adjust both LCD-bias supplies by varying the 2-kΩ potentiometer at the feedback node of the LT1611 regulator.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Figure 1



This LCD-bias supply provides better than 1% tracking of the positive and negative outputs.

Rolling-code generator uses flash µC

Wallace Ly, National Semiconductor Corp, Santa Clara, CA

MANY SECURITY-ALARM SYSTEMS require the use of a random number. A computer program uses this random number to create a sequence of random numbers to prevent unwanted visitors from gaining entry into a protected facility. You can use a "rolling-code generator" to produce random numbers. To implement such a generator, you would typically need a microcontroller with external memory. Instead, you can use National Semiconductor's COP8SBR flash

μC with "virtual-EEPROM" technology. This technology allows you to use a section of flash memory as if it were EEPROM. Because this μC is a true-flash device, the maximum number of erase/write cycles is typically 100,000 cycles. The flow chart in **Figure 1** and the C code in **Listing 1** show the adaptation of a textbook LFSR (linear-finite shift register) to the COP8 flash μC.

An initial "seed" first drives the input. The seed then traverses several exclusive-OR stages. The routine then saves the result to a virtual-EEPROM location. This approach allows an embedded-system designer to easily create a highly secure system without incurring the cost of an external nonvolatile memory, such as a dedicated serial EEPROM. You can download **Listing 1** from *EDN*'s Web site, www.ednmag.com. Click on "Search Databases," then enter the Software Center to download the file for Design Idea #2704. You can find additional information about the COP8SBR and its virtual-E² feature at www.national.com/cop8.

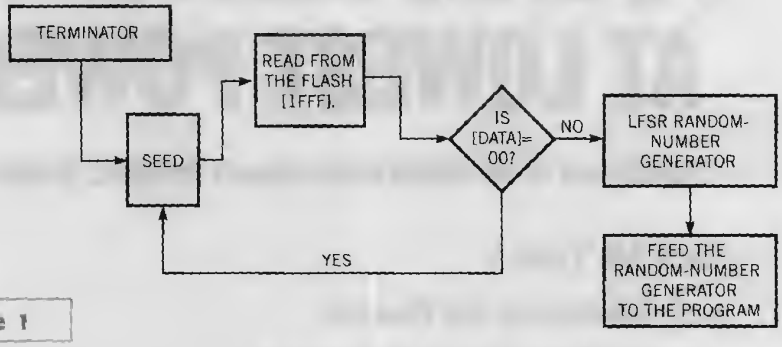


Figure 1

This random-number generator uses the virtual-E² feature of the COP8 μC.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

LISTING 1—SOURCE CODE FOR A VIRTUAL-EEPROM BASED RANDOM-NUMBER GENERATOR

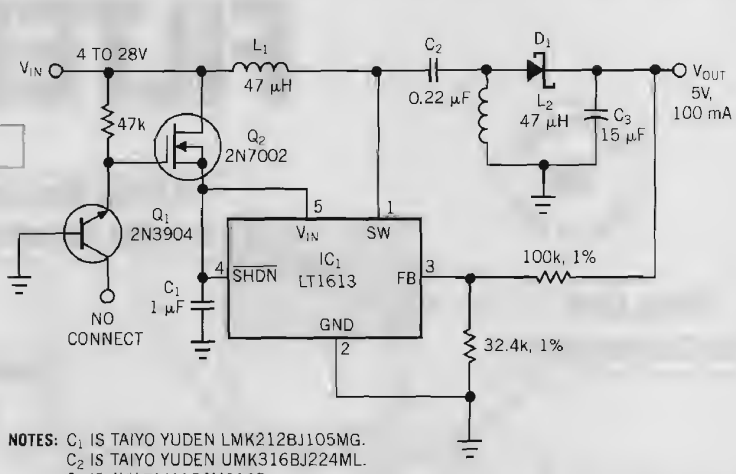
```
#include <8cbr.h>
#include <flash_op.h>


//***************************************************************
//
// Description: The following is a random number generator,
//              its implementation is the use of a 16 bit
//              LFSR, by XORing the correct bits we arrive
//              at a pseudo random number
//
// Implementation: The pseudo number is created by using the
//                 feedback equation (for a 16 bit word)
//                 bit15=bit5 XOR bit4 XOR bit3 XOR bit0
//
//***************************************************************

void random(){

        unsigned int seed_upper;  // The seed variables
        unsigned int seed_lower;

        // Seed the random number generator
        // Choose any random numbers for the MSB and
        // LSB of the initial random numbers

        readbf(0x1FFF);     // Do a read at the high location
        seed_upper=ISPRD;

        readbf(0x2000);     // Do a read at the low location
        seed_lower=ISPRD;

        if (!seed_upper && !seed_lower){       // If it is zero then continue

          }

        else {

          seed_upper=10;  // Otherwise give it a seed
          seed_lower=20;

          }

        bits temp1;             // Some Xoring Bits
        bits temp2;


        char flag;
        bits carry;

        temp1=(bits)seed_upper;             // Assign the upper & lower
        temp2=(bits)seed_lower;

        carry.0=temp2.5^temp2.4;     // Now do the Xoring
        carry.0=carry.0^temp2.3;
        carry.0=carry.0^temp2.0;

        if (temp2.0)
            flag=1;

            else flag=0;

            temp1=temp1>>1;                         // Bit shift left
            temp2=temp2>>1;


            if (flag)
                temp2.7=1;


            temp1.7=carry.0;                        // Use the Carry Bit


            seed_upper=temp1;
            seed_lower=temp2;

            page_erase(0x1FFF).          // Erase the location
            cwritebf(0x1FFF,seed_upper);       // Save the upper byte
            cwritebf(0x2000,seed_lower);       // Save the lower byte

} // end of the random routine


void main(){

        unsigned int i;             // A counter variable
        unsigned int random_arr[2]; // A random number array

                // Set the clock timer
                FGMTTM=0x7B;


                                        // copy it back to the random numbers
                readbf(0x1FFF);    // Read the result
                random_arr[0]=ISPRD;      // From the ISPRD register

                readbf(0x2000);
                random_arr[1]=ISPRD;      // Both the Upper and Lower
                                // Seed
        while (1){


                page_erase(0x1FFF);             // Page Erase

                cwritebf(0x1FFE,random_arr[0]);  // Write the byte back
                cwritebf(0x2000,random_arr[1]);  // Write the byte back


                // Get a number randomly generated through 100 iterations
                for (i=0;i<100;i++)             // Call the random number
                     {                          // generator a hundred times
                      random();
                }

                                // Application code goes over here

                   }


} // end of main
```

SEPIC generates 5V at 100 mA

Dongyan Zhou, Linear Technology Corp, Milpitas, CA

Some applications require an input voltage higher than the breakdown voltage of the IC supply pin. In boost converters and SEPICs (single-ended primary-inductance converters), you can separate the V_{IN} pin of the IC from the input inductor and use a simple zener regulator to generate the supply voltage for the IC. **Figure 1** shows a SEPIC that takes a 4 to 28V input and generates 5V at 100 mA.

Figure 1



NOTES: C_1 IS TAIYO YUDEN LMK212BJ105MG.
C_2 IS TAIYO YUDEN UMK316BJ224ML.
C_3 IS AVX TAJA156M010R.
L_1 AND L_2 ARE MURATA LQH3C470K34.
D_1 IS MOTOROLA MBR0540T3.

Q_1 stands in for a zener diode in this SEPIC with a wide input-voltage range.

In this application, Q_1 and Q_2 generate the supply voltage for IC_1 because the supply voltage exceeds IC_1's maximum input voltage. The circuit uses Q_1 in place of a zener diode to save cost. The emitter-to-base breakdown voltage gives a stable 6V reference. The follower, Q_2, provides the supply voltage for the IC. This circuit demonstrates an inexpensive way to extend the input range of the IC.

This SEPIC can step up or step down the input voltage. Because the flying capacitor, C_2, breaks the input-to-output dc path, the output disconnects from the input when you shut down the device, inhibiting any possible load current in shutdown mode, which is important for portable applications and which prevents the input voltage from appearing at the output.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

Look-up table helps bit flipping

Brad Bierschenk, High End Systems, Austin, TX

In certain instances in embedded software, a programmer needs to flip the order of the bits in a byte so that B_7 to B_0 become B_0 to B_7. Bit flipping is useful, for example, when a synchronous serial port does not allow programmatic selection of bit order, such as MSB first or LSB first, for its shift register. You need a software method to translate data if the processor sends data to a receiving device that expects a certain bit order, but the serial port can provide only the other bit order.

One solution is to provide a look-up table in ROM in which the value of each byte in the table is its offset into the table but with a reversed bit order. In other words, the first byte is offset 0 (00000000b), the second byte is offset 1 (10000000b), the third byte is offset 2 (01000000b), and so on. The program needs only to load the value to be translated into a register that you can use as an offset, index the look-up table, and load the corresponding value from the index+offset location (**Listing 1**). Using the Philips 8×C51 architecture as an example, you can use the 16-bit DPTR (data pointer) plus an 8-bit offset in accumulator (A) to load the accumulator with a byte value.

LISTING 1—BIT-FLIPPING CODE SEGMENT

```
CodeSegment:
        ; Load value into accumulator (hex AA)
        mov         A, #10101010b
        ; Load lookup table address into index
        mov     DPTR,#InvertTable
        ; Load "flipped" value into accumulator
        movc    A,@A+DPTR
        ; Accumulator should now hold hex 55
```

This approach is dynamically more efficient than rotating a byte location through carry bits, but it is not the most statically efficient approach because the look-up table requires 256 bytes of ROM. You can download the inversion table from *EDN*'s Web site, www.ednmag.com. Click on "Search Databases" and then enter the Software Center to download the file for Design Idea #2621.

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.